САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OpenMP»

Выполнил(а): Шеметов Алексей Игоревич

Номер ИСУ: 338978

студ. гр. М3134

Санкт-Петербург

2021

Что такое OpenMP?

OpenMP – это библиотека, которая позволяет легко распараллелить простые участки кода. Параллельное программирование применяется тогда, когда для последовательных блоков программ требуется уменьшить время ее выполнения.

Программная модель OpenMP

Основной поток порождает дочерние потоки по мере необходимости. Программирование путем вставки директив компилятора в ключевые места исходного кода программы. Компилятор интерпретирует эти директивы и вставляет в соответствующие места программы библиотечные вызовы для распараллеливания участков кода.

Директива #pragma omp parallel for указывает на то, что данный цикл следует разделить по итерациям между потоками. Количество потоков можно контролировать из программы, командой

omp_set_num_threads(<num_threads>);

Для компиляции программы следует указать ключ -openmp

В OpenMP есть команда schedule(type, [, block_size]), где type – принимает значения: static, dynamic, guided и runtime.

static – итерации равномерно распределяются по потокам. Т.е. если в цикле 1000 итераций и 4 потока, то один поток обрабатывает все итерации с 1 по 250, второй – с 251 по 500, третий - с 501 по 750, четвертый с 751 по 1000. Если при этом задан еще и размер блока, то все итерации блоками заданного размера циклически распределяются между потоками. Статическое распределение работы эффективно, когда время выполнения итераций равно, или приблизительно равно. Если это не так, то разумно использовать следующий тип распределения работ.

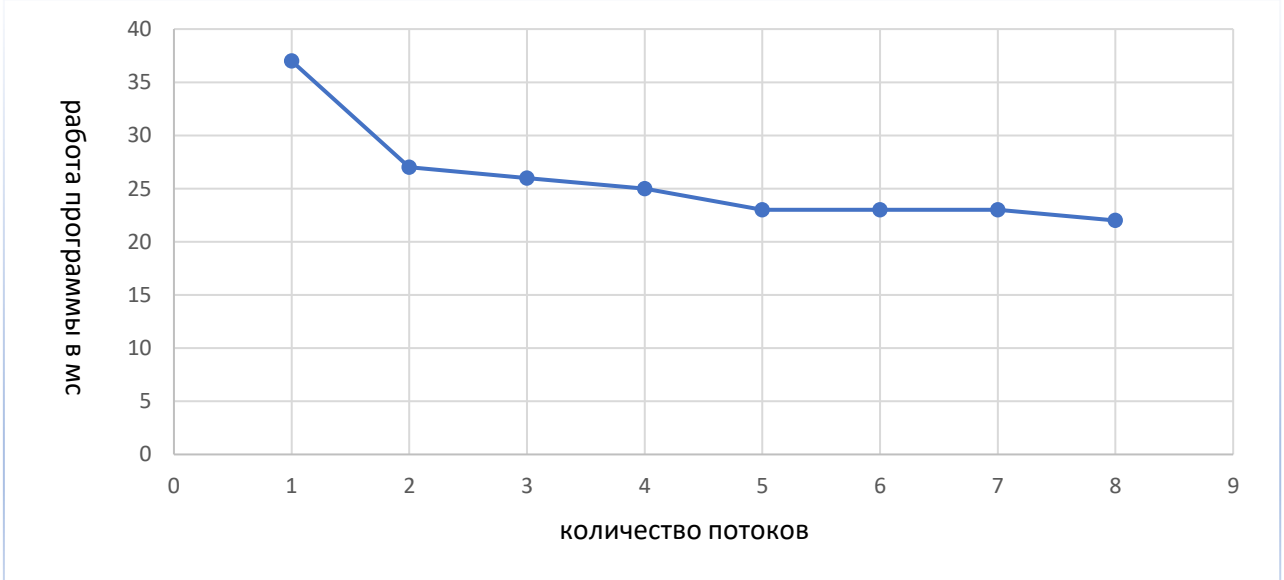
dynamic – работа распределяется пакетами заданного размера (по умолчанию размер равен 1) между потоками. Как только какой-либо из потоков заканчивает обработку своей порции данных, он захватывает следующую. Стоит отметить, что при этом подходе несколько большие накладные расходы, но можно добиться лучшей балансировки загрузки между потоками.

guided – данный тип распределения работы аналогичен предыдущему, за тем исключением, сто размер блока изменяется динамически в зависимости от того, сколько необработанных итераций осталось. Размер блока постепенно уменьшается вплоть до указанного значения. При таком подходе можно достичь хорошей балансировки при меньших накладных расходах.

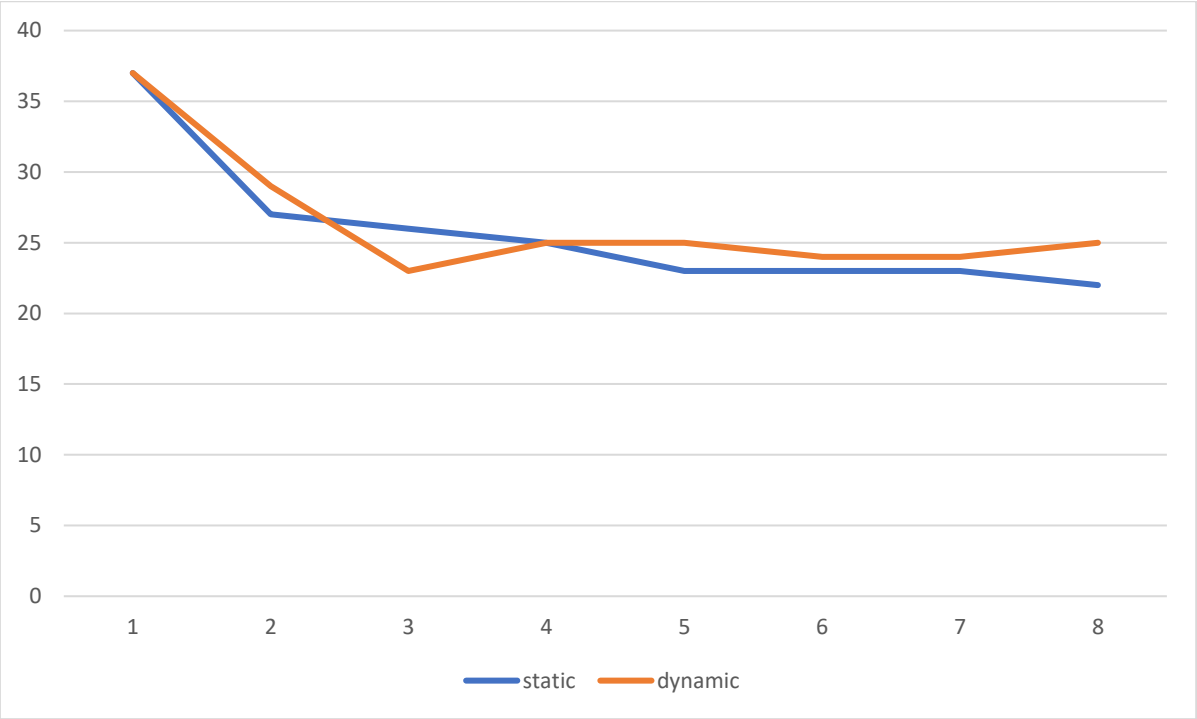
runtime – тип распределения определяется в момент выполнения программы. Это удобно в экспериментальных целях для выбора оптимального значения типа и размера блока.

Реализация алгоритма (модификация Hard, автоматическая контрастность изображения)

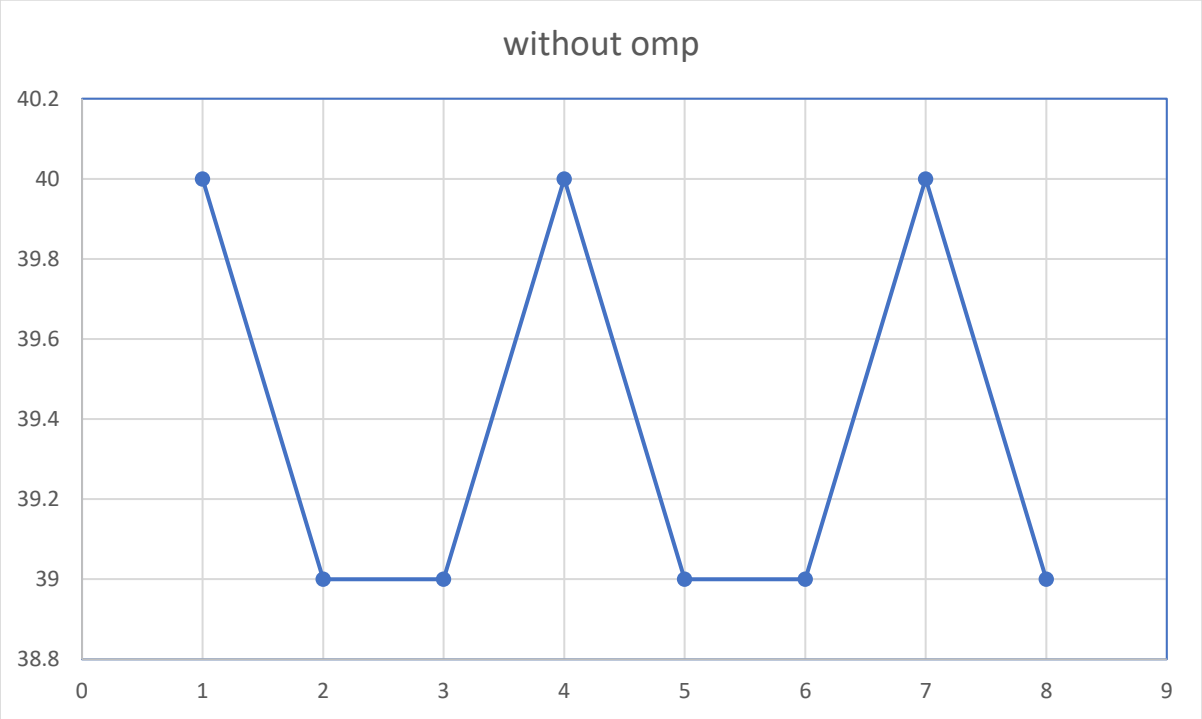
На некоторых фотографиях самые темные пиксели не равны 0, а самые яркие 255. Из-за этого некоторые объекты картинки трудно различимы. Для того, чтобы текущий диапазон “растянуть” на [0;255] найдем пиксель с минимальной яркостью и пиксель с максимальной яркостью (если kf != 0, то пропускаем пиксели до тех пор, пока skipPixel / totalPixel < kf, затем ищем пиксель с максимальной и минимальной яркостью). После нахождения мин и макс изменяем каждый пиксель по формуле curPixel = colorRange * (curPixel - min) / (max - min); (colorRange – цветовой диапазон (если, например, самый яркий пиксель должен быть меньше 255)). Данный процесс можно распараллелить, так как никакой пиксель не зависит от другого. Далее записываем пиксели в том же порядке при считывании в заданный файл.



«Рисунок №1 – график работы программы при разном числе потоков и при одинаковом параметре schedule»



«Рисунок №2 – сравнение работы программы при одинаковом количестве потоков и при разных параметрах schedul»



«Рисунок №3 – время работы программы при выключенном OpenMP (используется один поток)»

Листинг

Компилятор: Visual C++ 14.2

PNM.h

```
#include <omp.h>
#include <fstream>

class PNM
{
public:
	PNM(std::string file, double kf, int threads);
	void write(std::string file);
	void increaseContrast();
private:
	std::string version;
	int width, height;
	int colorRange;
	double kf;
	int threads;
	int Ncolor;
	unsigned char*** data;
};
```

PNM.cpp

```
#include "PNM.h"

PNM::PNM(std::string file, double kf, int threads) : kf(kf), threads(threads)
{
	std::ifstream img(file, std::ios::binary);

	img >> version;
	if (version == "P5") {
		Ncolor = 1;
	}
	else {
		Ncolor = 3;
	}

	img >> width >> height >> colorRange;
	char* buf = new char[width * Ncolor];
	img.read(buf, 1);

	data = new unsigned char** [height];
	for (int row = 0; row < height; row++) {
		data[row] = new unsigned char* [width];
		for (int col = 0; col < width; col++) {
			data[row][col] = new unsigned char[Ncolor];
		}
	}

	for (int row = 0; row < height; row++) {
		img.read(buf, width * Ncolor);
```

```
            for (int col = 0; col < width; col++) {
                  for (int colors = 0; colors < Ncolor; colors++) {
                        data[row][col][colors] = buf[col * Ncolor + colors];
                  }
            }
      }


      img.close();
}

void PNM::increaseContrast() {
      omp_set_num_threads(threads);
      unsigned char minColor = 255, maxColor = 0;

      int* arr = new int[colorRange + 1];
      for (int i = 0; i < colorRange + 1; i++) {
            arr[i] = 0;
      }

#pragma omp parallel for schedule(static)
      for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                  for (int colors = 0; colors < Ncolor; colors++) {
                        arr[int(data[row][col][colors])]++;
                  }
            }
      }

      if (kf - 1e-6 > 0) {
            int pixels = width * height * 3;
            int count = 0, i;

            for (i = 0; count < pixels * kf; i++) {
                  count += arr[i];
            }
            minColor = i;

            count = 0;
            for (i = colorRange; count < pixels * kf; i--) {
                  count += arr[i];
            }
            maxColor = i;
      }
      else {
            int i;
            for (i = 0; arr[i] == 0; i++) {}
            minColor = i;

            for (i = colorRange; arr[i] == 0; i--) {}
            maxColor = i;
      }

      if (maxColor == minColor) {
            return;
      }
```

```
#pragma omp parallel for schedule(static)
	for (int row = 0; row < height; row++) {
		for (int col = 0; col < width; col++) {
			for (int colors = 0; colors < Ncolor; colors++) {
				int newPixel = colorRange * (data[row][col][colors] -
minColor) / (maxColor - minColor);
				if (newPixel > 0 && newPixel < colorRange + 1) {
					data[row][col][colors] = newPixel;
				}
				else if (newPixel <= 0) {
					data[row][col][colors] = 0;
				}
				else {
					data[row][col][colors] = 255;
				}
			}
		}
	}
}

void PNM::write(std::string file) {
	std::ofstream img(file, std::ios::binary);

	img << version + "\n" << width << " " << height << "\n" << colorRange
<< "\n";

	char* buf = new char[width * Ncolor];
	for (int row = 0; row < height; row++) {
		for (int col = 0; col < width; col++) {
			for (int colors = 0; colors < Ncolor; colors++) {
				buf[col * Ncolor + colors] = data[row][col][colors];
			}
		}
		img.write(buf, width * Ncolor);
	}

	delete data;
	img.close();
}
```

Main.cpp

```
#include "PNM.h"
#include <iostream>
#include <ctime>
#include <string>

int main(int argc, char** argv) {
	if (argc < 4) {
		std::cerr << "no arguments!";
		return -1;
	}

	int threads = std::atoi(argv[1]);
	std::string fileInput = argv[2];
	std::string fileOutput = argv[3];
```

```
	double kf = 0;
	if (argc >= 5) {
		kf = std::stod(argv[4]);
	}

	try {
		if (kf >= 0.5) {
			throw "too high value of kf";
		}
		if (kf < 0) {
			throw "kf cannot be negative";
		}
	}
	catch (char* ch) {
		std::cerr << ch;
		return -1;
	}

	std::ifstream fileIn(fileInput);
	if (!fileIn.is_open()) {
		std::cerr << "file not found";
		return -1;
	}
	std::string version;
	fileIn >> version;
	fileIn.close();
	if (version != "P5" && version != "P6") {
		std::cerr << "unsoported version";
		return -1;
	}

	unsigned int start = clock();
	PNM pnm(fileInput, kf, threads);
	unsigned int startWork = clock();
	pnm.increaseContrast();
	unsigned int endWork = clock();
	pnm.write(fileOutput);
	unsigned int end = clock();
	std::cout << "Reading: " << startWork - start << "ms\n";
	std::cout << "Time work: " << endWork - startWork << "ms\n";
	std::cout << "Writing: " << end - endWork << "ms\n";
	std::cout << "Total: " << end - start << "ms";

	return 0;
}
```